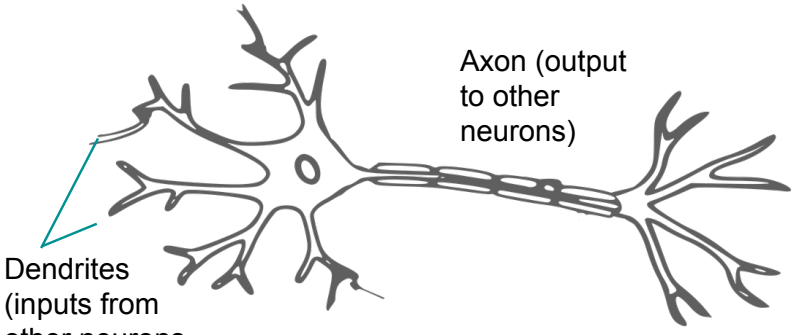
10-701 Introduction to Machine Learning (PhD)
Lecture 9: Neural Networks

Leila Wehbe
Carnegie Mellon University
Machine Learning Department

Slides based on on Tom Mitchell's
10-701 Spring 2016 material
Readings: Hal Daumé III Chapter 4,10
[TM] Ch. 4, [CB] Ch. 5

The Perceptron

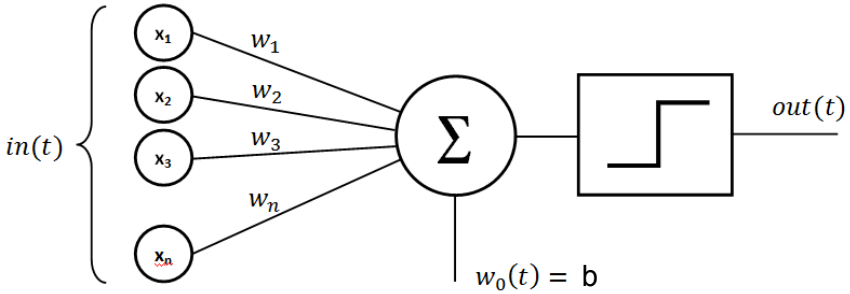
Inspired by biological neurons



Axon (output to other neurons)

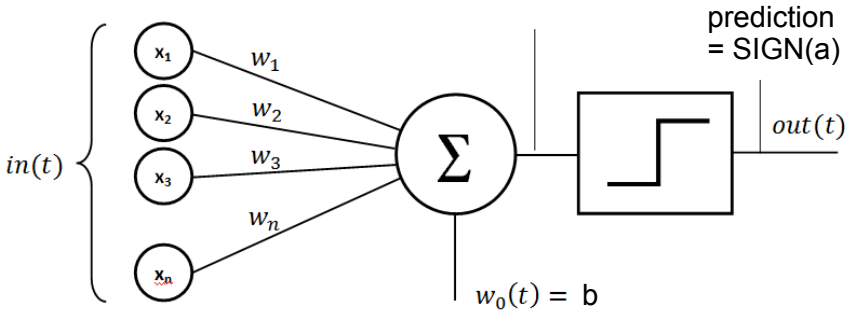
Dendrites (inputs from other neurons, can be excitatory or inhibitory)

Perceptron



Perceptron

$$a = b + \sum_{d=1}^{D} w_d x_d$$



Error driven learning

$$a = b + \sum_{d=1}^{D} w_d x_d$$

- At each step, return SIGN(a)
- if SIGN(a)≠ y update parameter
- otherwise don't change

Algorithm 5 PERCEPTRONTRAIN(**D**, *MaxIter*)

```
w_d ← 0, for all d = 1...D                    // initialize weights
b ← 0                                         // initialize bias
for iter = 1...MaxIter do
    for all (x,y) ∈ D do
        a ← Σ_{d=1}^{D} w_d x_d + b           // compute activation for this example
        if ya ≤ 0 then
            w_d ← w_d + yx_d, for all d = 1...D   // update weights
            b ← b + y                         // update bias
        end if
    end for
end for
return w_0, w_1, ..., w_D, b
```

from http://ciml.info/dl/v0_99/ciml-v0_99-ch08.pdf

Does this move a in the right direction?

- update **w' = w +yx = w + x**
- b' = b + y = b+1

$$\begin{aligned} a' &= \sum_{d=1}^{D} w'_d x_d + b' \\ &= \sum_{d=1}^{D} (w_d + x_d) x_d + (b+1) \\ &= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\ &= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a \end{aligned}$$

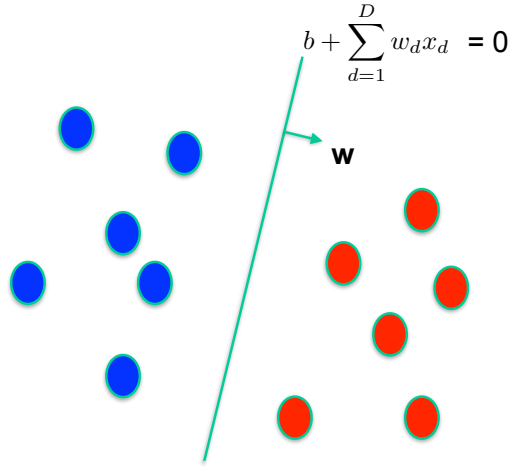
Does this move a in the right direction?

- update **w' = w +yx = w + x**
- b' = b + y = b+1

$$
\begin{aligned}
a' &= \sum_{d=1}^{D} w'_d x_d + b' \\
&= \sum_{d=1}^{D} (w_d + x_d) x_d + (b+1) \\
&= \sum_{d=1}^{D} w_d x_d + b + \sum_{d=1}^{D} x_d x_d + 1 \\
&= a + \sum_{d=1}^{D} x_d^2 + 1 \quad > \quad a
\end{aligned}
$$

a becomes more positive
(not guaranteed that a>0)

What is the decision boundary?



How good is this algorithm?

- Convergence: an entire pass without changing the weights.
- If the data is linearly separable, the algorithm will converge. But not necessarily to the "best" boundary

Notion of margin

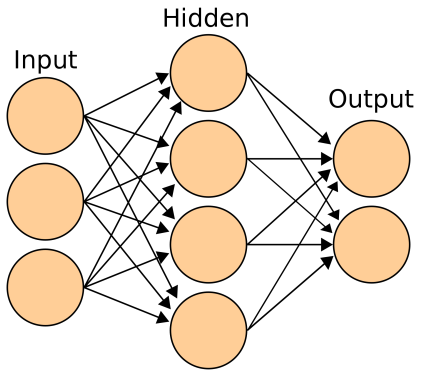
$$
margin(\mathbf{D}, \boldsymbol{w}, b) = \begin{cases} \min_{(\boldsymbol{x},y)\in\mathbf{D}} y(\boldsymbol{w}\cdot\boldsymbol{x}+b) & \text{if } \boldsymbol{w} \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases}
$$

$$
margin(\mathbf{D}) = \sup_{\boldsymbol{w},b} margin(\mathbf{D}, \boldsymbol{w}, b)
$$

If data is linearly separable with margin γ and $||\mathbf{x}|| \leq 1$, then algorithm will converge in $\frac{1}{\gamma^2}$ updates

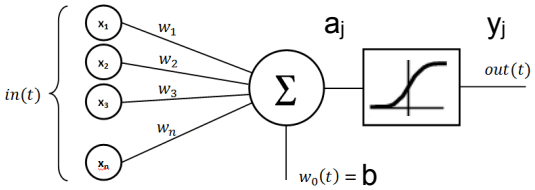
Neural Networks

Every node is analogous to a neuron

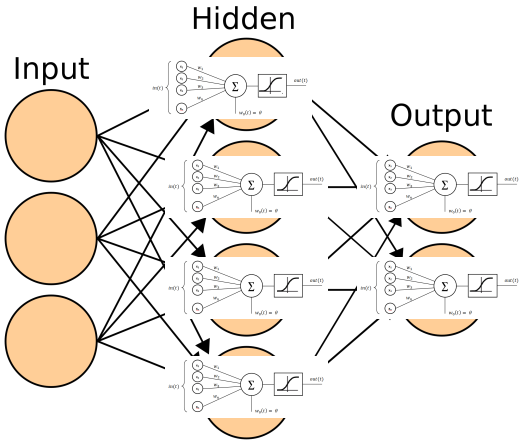


Every node is analogous to a neuron

sigmoid unit

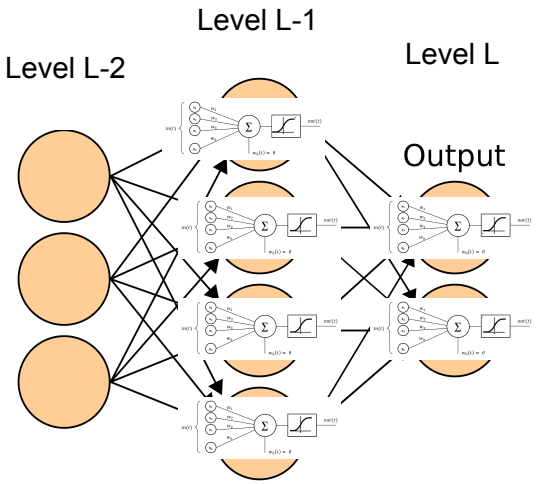


Every node is analogous to a neuron



Prediction:
Forward Propagation

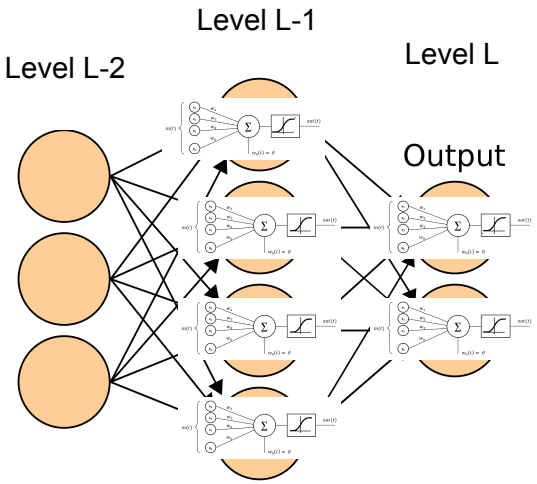
$$y^j_{L-1} = \sigma\left(\sum_k w^{jk}_{L-1} y^k_{L-2} + b^j_{L-1}\right)$$



Prediction:
Forward Propagation

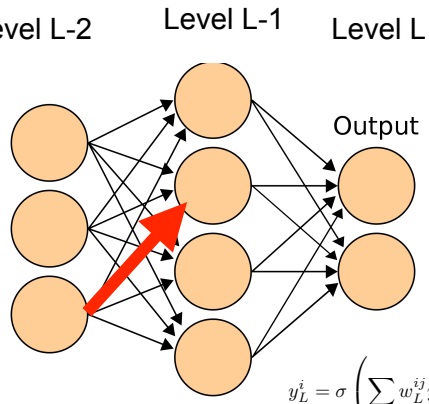
$$y^i_L = \sigma\left(\sum_j w^{ij}_L y^j_{L-1} + b^i_L\right)$$

$$\sigma\left(\sum_j w^{ij}_L \sigma\left(\sum_k w^{jk}_{L-1} y^k_{L-2} + b^j_{L-1}\right) + b^i_L\right)$$



How to train?

What is $\dfrac{\partial E}{\partial w^{23}_{L-1}}$?

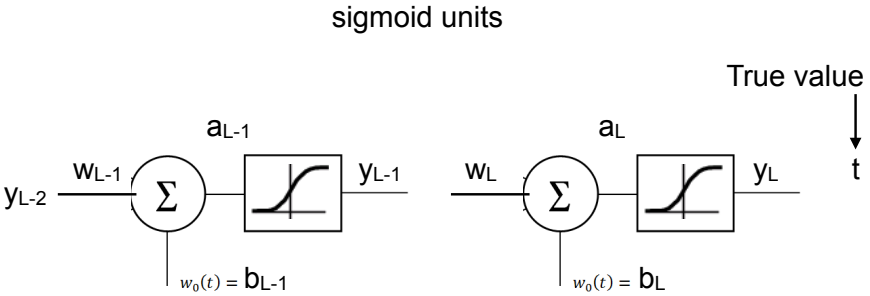


$$E = \frac{1}{2}\sum_i (y^i_L - t^i)^2$$

$$y^i_L = \sigma\left(\sum_j w^{ij}_L y^j_{L-1} + b^i_L\right)$$

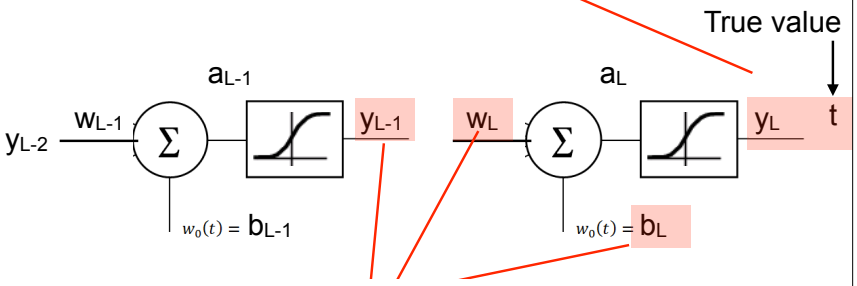
$$\sigma\left(\sum_i w^{ij}_L \sigma\left(\sum_k w^{jk}_{L-1} y^k_{L-2}\right) + b^j_{L-1}\right)$$

Backprop with one node per layer



Backprop with one node per layer

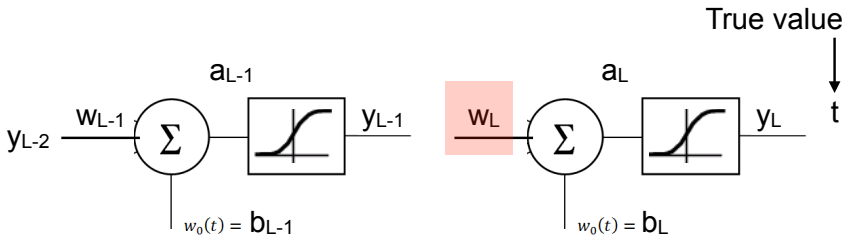
Loss function $E = \frac{1}{2}(y_L - t)^2$



Find the gradient for one training point at earlier nodes and parameters: how much does a very small change in the value of nodes and parameters affect the loss for that point?

Backprop with one node per layer

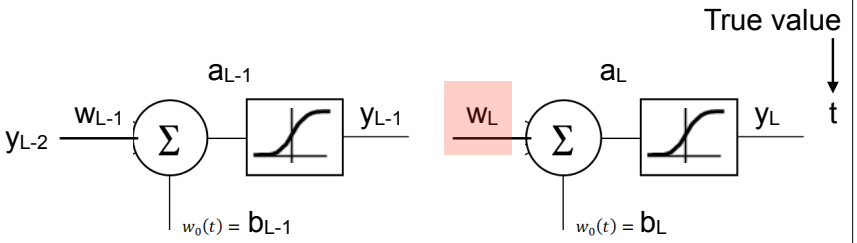
Loss function $E = \frac{1}{2}(y_L - t)^2$



$$\frac{\partial E}{\partial w_L}$$

Backprop with one node per layer

Loss function $E = \frac{1}{2}(y_L - t)^2$



$$\frac{\partial E}{\partial w_L}$$

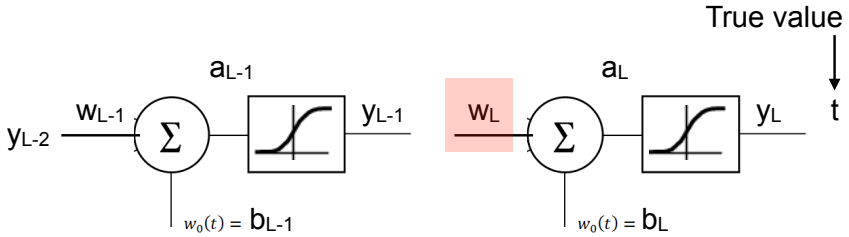
$E = \frac{1}{2}(y_L - t)^2$

$y_L = \sigma(a_L)$

$a_L = w_L y_{L-1} + b_L$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$



$$\frac{\partial E}{\partial w_L} = \frac{\partial a_L}{\partial w_L}\frac{\partial y_L}{\partial a_L}\frac{\partial E}{\partial y_L}$$

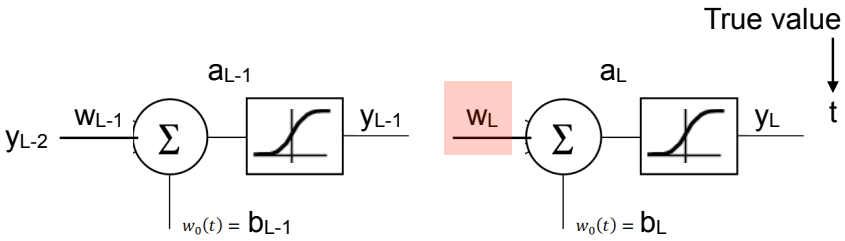
$E = \frac{1}{2}(y_L - t)^2$

$y_L = \sigma(a_L)$

$a_L = w_L y_{L-1} + b_L$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$

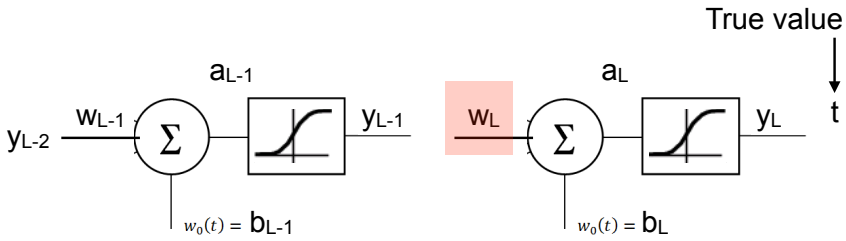


$$\frac{\partial E}{\partial w_L} = \frac{\partial a_L}{\partial w_L}\frac{\partial y_L}{\partial a_L}\frac{\partial E}{\partial y_L}$$

$$E = \frac{1}{2}(y_L - t)^2$$
$$y_L = \sigma(a_L)$$
$$a_L = w_L y_{L-1} + b_L$$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$

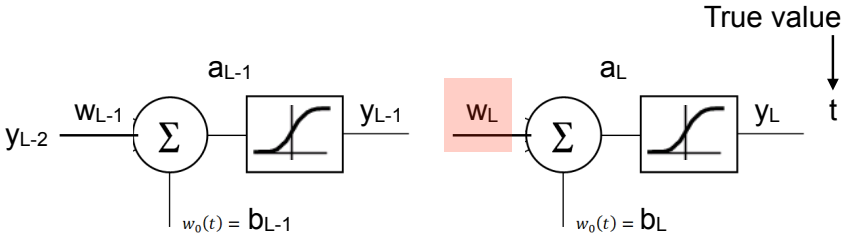


$$\frac{\partial E}{\partial w_L} = y_{L-1}\sigma'(a_L)(y_L - t)$$

$$E = \frac{1}{2}(y_L - t)^2$$
$$y_L = \sigma(a_L)$$
$$a_L = w_L y_{L-1} + b_L$$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$

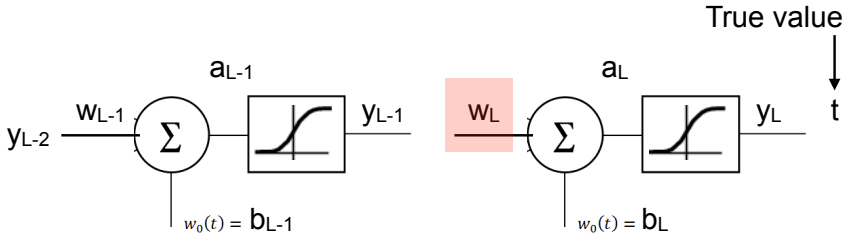


$$\frac{\partial E}{\partial w_L} = y_{L-1}\sigma'(a_L)(y_L - t)$$

$$\sigma'(a_L) = \sigma(a_L)(1 - \sigma(a_L))$$
$$\sigma(a_L) = y_L$$

Backprop with one node per layer

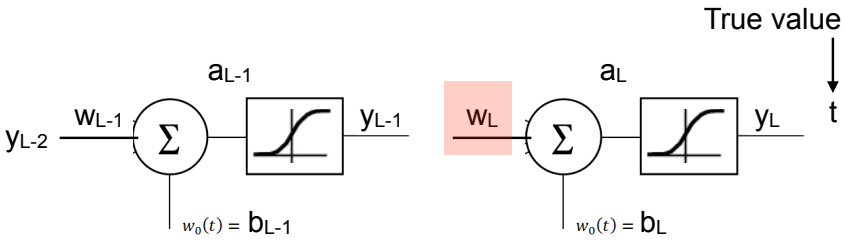
$$E = \frac{1}{2}(y_L - t)^2$$



$$\frac{\partial E}{\partial w_L} = y_{L-1}y_L(1 - y_L)(y_L - t)$$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$

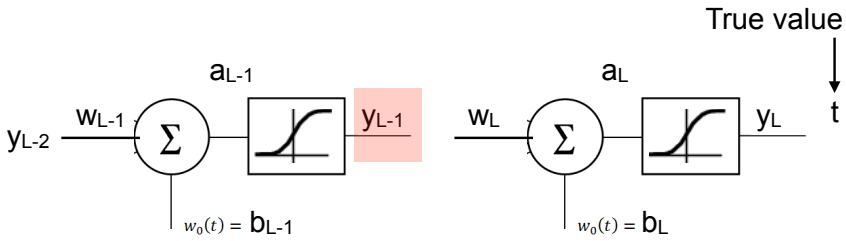


Effect of ∂w_L depends on y_{L-1}

$$\frac{\partial E}{\partial w_L} = y_{L-1} y_L (1 - y_L)(y_L - t)$$

Backprop with one node per layer

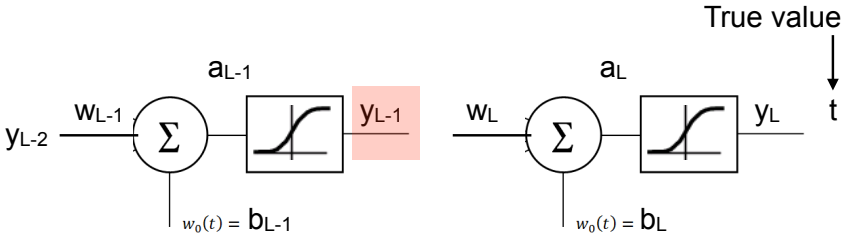
$$E = \frac{1}{2}(y_L - t)^2$$



$$\frac{\partial E}{\partial y_{L-1}} = \frac{\partial a_L}{\partial y_{L-1}} \frac{\partial y_L}{\partial a_L} \frac{\partial E}{\partial y_L}$$

Backprop with one node per layer

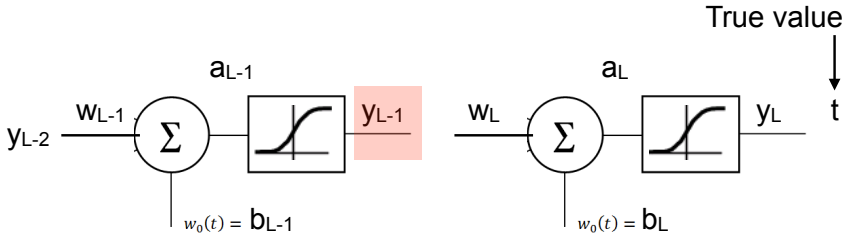
$$E = \frac{1}{2}(y_L - t)^2$$



$$\frac{\partial E}{\partial y_{L-1}} = \frac{\partial a_L}{\partial y_{L-1}} \frac{\partial y_L}{\partial a_L} \frac{\partial E}{\partial y_L} \qquad a_L = w_L y_{L-1} + b_L$$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$

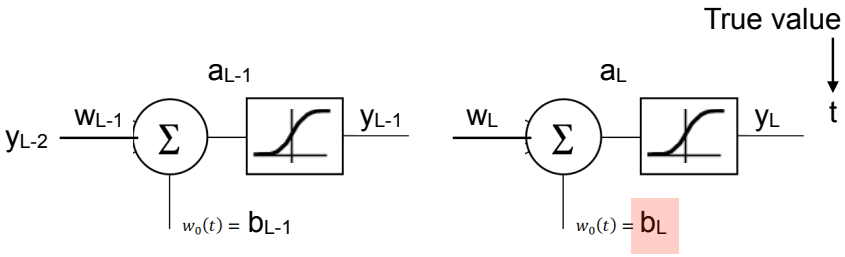


$$a_L = w_L y_{L-1} + b_L$$

$$\frac{\partial E}{\partial y_{L-1}} = w_L y_L (1 - y_L)(y_L - t)$$

Backprop with one node per layer

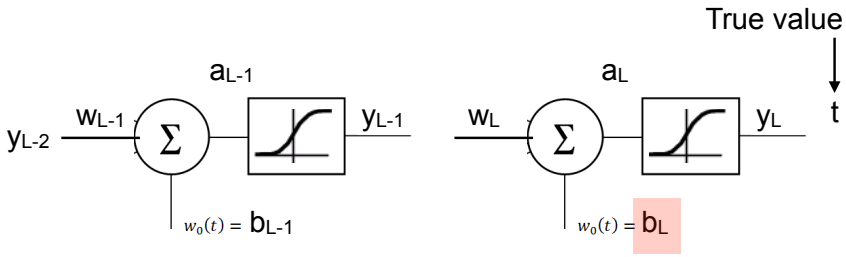
$$E = \frac{1}{2}(y_L - t)^2$$



$$\frac{\partial E}{\partial b_L} = \frac{\partial a_L}{\partial b_L}\frac{\partial y_L}{\partial a_L}\frac{\partial E}{\partial y_L}$$

Backprop with one node per layer

$$E = \frac{1}{2}(y_L - t)^2$$

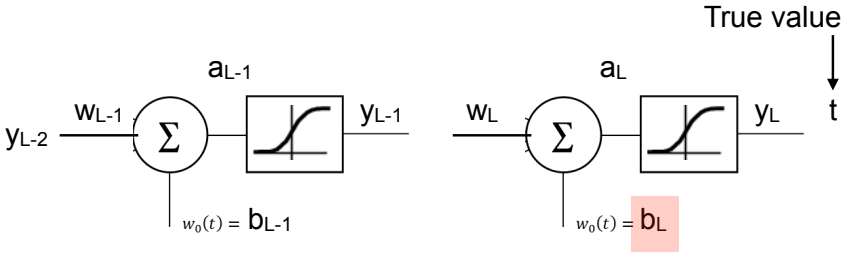


$$\frac{\partial E}{\partial b_L} = \frac{\partial a_L}{\partial b_L}\frac{\partial y_L}{\partial a_L}\frac{\partial E}{\partial y_L}$$

$$a_L = w_L y_{L-1} + b_L$$

Backprop with one node per layer

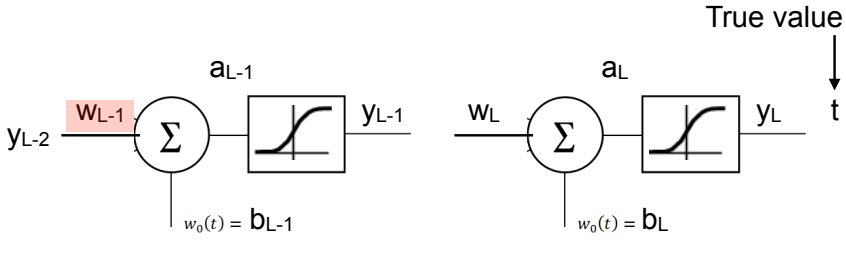
$$E = \frac{1}{2}(y_L - t)^2$$



$$\frac{\partial E}{\partial b_L} = 1 y_L(1 - y_L)(y_L - t)$$

Backprop with one node per layer

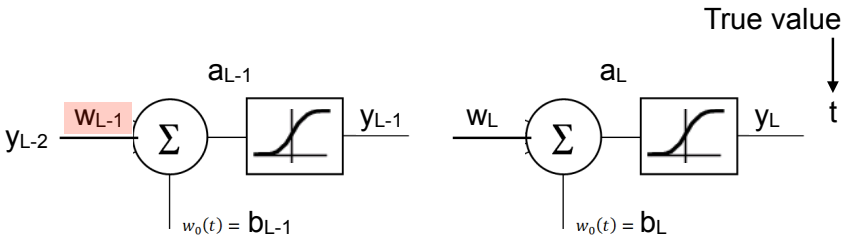
Loss function $E = \frac{1}{2}(y_L - t)^2$



$$\frac{\partial E}{\partial w_{L-1}}$$

Backprop with one node per layer

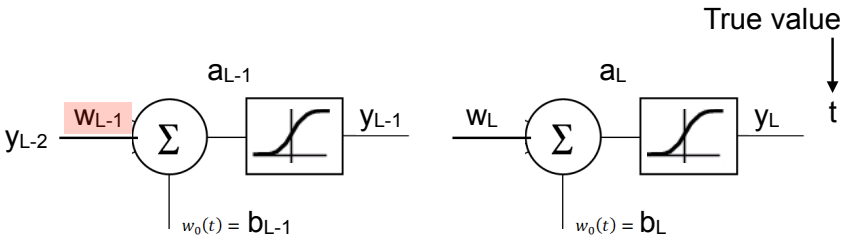
Loss function $E = \frac{1}{2}(y_L - t)^2$



$$\frac{\partial E}{\partial w_{L-1}} = \frac{\partial a_{L-1}}{\partial w_{L-1}} \frac{\partial y_{L-1}}{\partial a_{L-1}} \frac{\partial E}{\partial y_{L-1}}$$

Backprop with one node per layer

Loss function $E = \frac{1}{2}(y_L - t)^2$



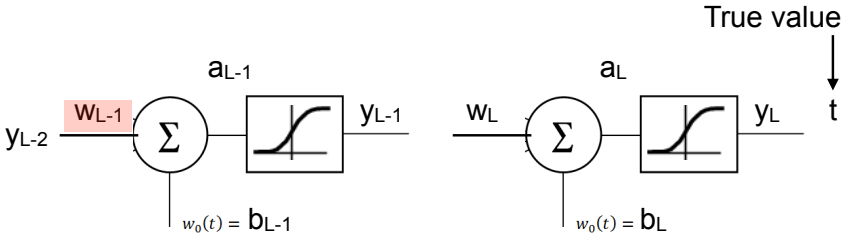
$a_{L-1} = w_{L-1} y_{L-2} + b_{L-1}$ $\quad y_{L-1} = \sigma(a_{L-1})$

$$\frac{\partial E}{\partial w_{L-1}} = \frac{\partial a_{L-1}}{\partial w_{L-1}} \frac{\partial y_{L-1}}{\partial a_{L-1}} \frac{\partial E}{\partial y_{L-1}}$$

Already Computed

Backprop with one node per layer

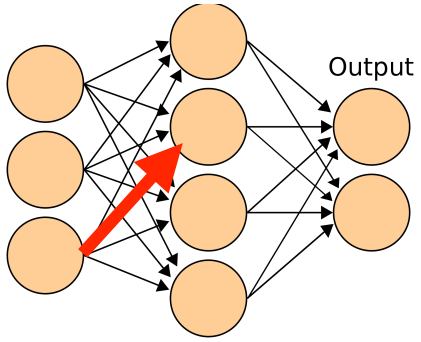
Loss function $E = \frac{1}{2}(y_L - t)^2$



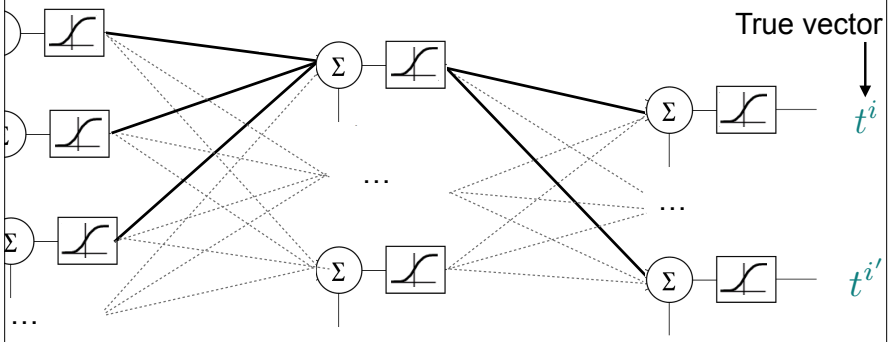
$$\frac{\partial E}{\partial w_{L-1}} = y_{L-2} y_{L-1}(1 - y_{L-1}) w_L y_L (1 - y_L)(y_L - t)$$

Backprop with more than one node per layer

What is $\frac{\partial E}{\partial w_{L-1}^{23}}$?

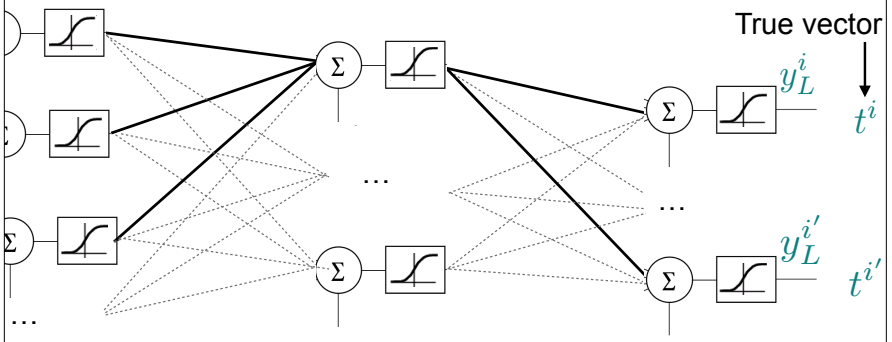


Backprop with more than one node per layer



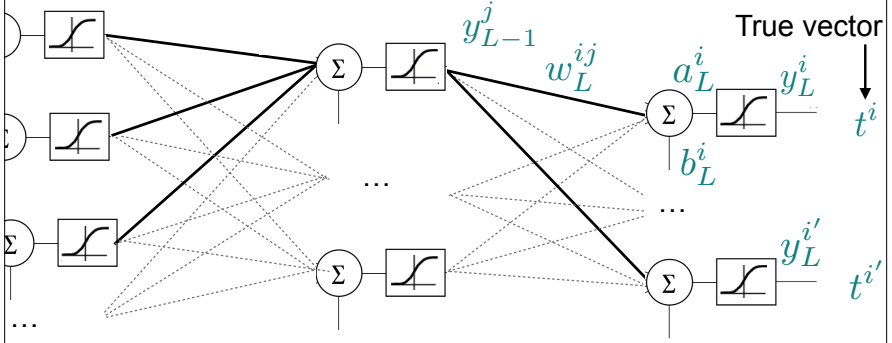
Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



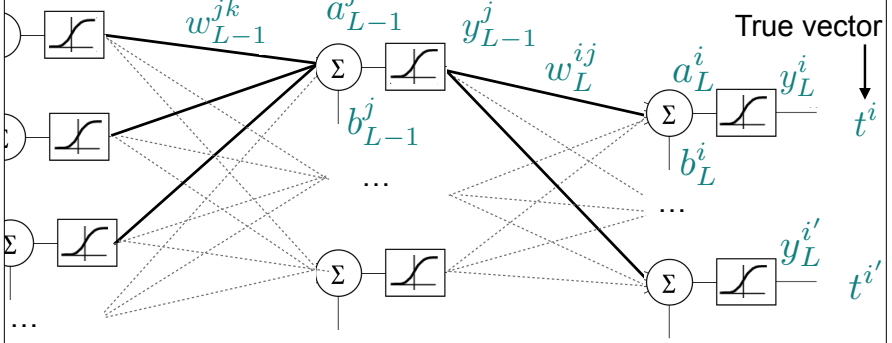
Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



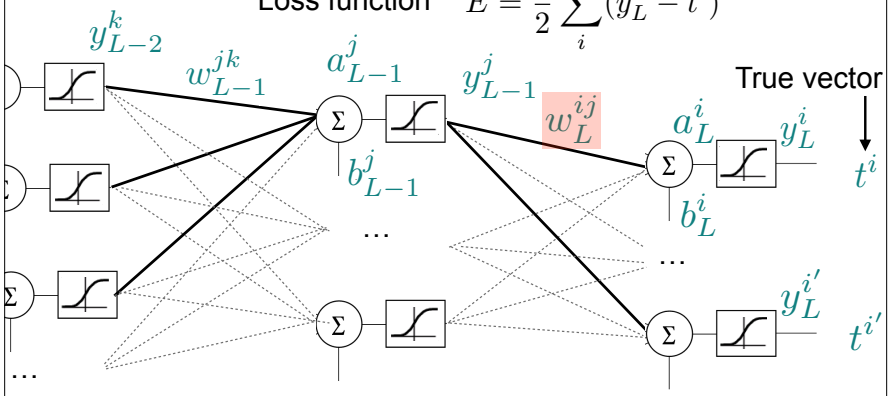
Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



Backprop with more than one node per layer

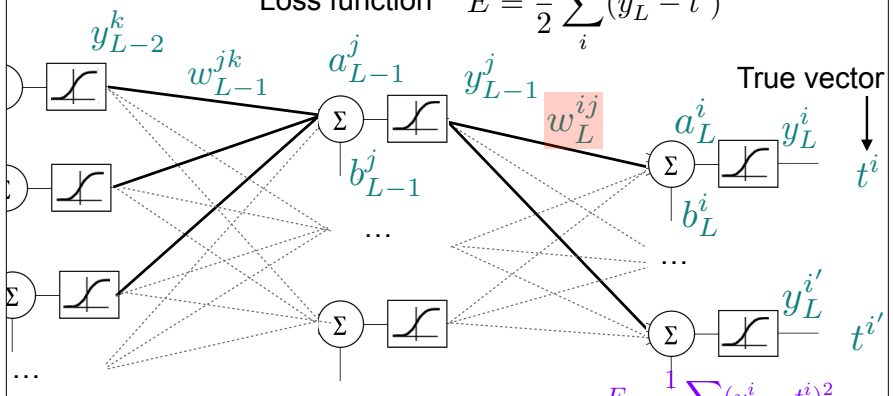
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial w_L^{ij}}$$

Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial w_L^{ij}} = \frac{\partial a_L^i}{\partial w_L^{ij}} \frac{\partial y_L^i}{\partial a_L^i} \frac{\partial E}{\partial y_L^i}$$

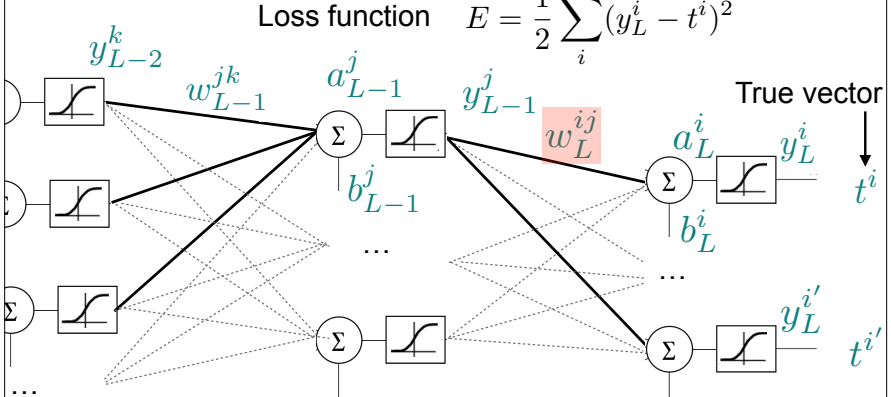
$$E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$$

$$y_L^i = \sigma(a_L^i)$$

$$a_L^i = \sum_j w_L^{ij} y_{L-1}^j + b_L^i$$

Backprop with more than one node per layer

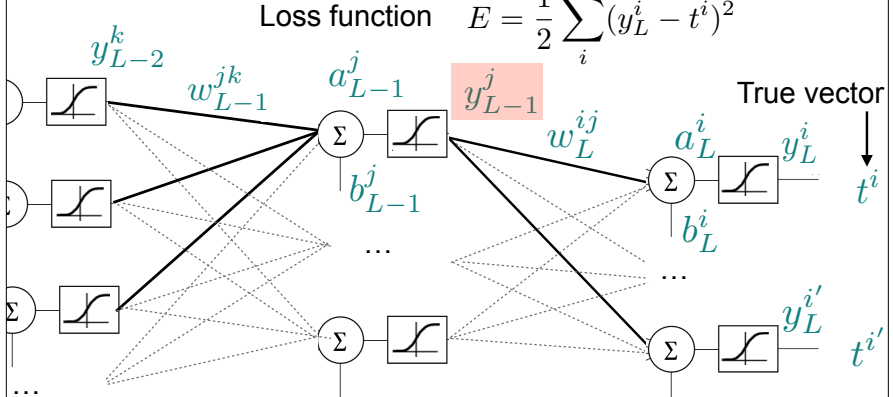
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial w_L^{ij}} = y_{L-1}^j y_L^i (1 - y_L^i)(y_L^i - t^i)$$

Backprop with more than one node per layer

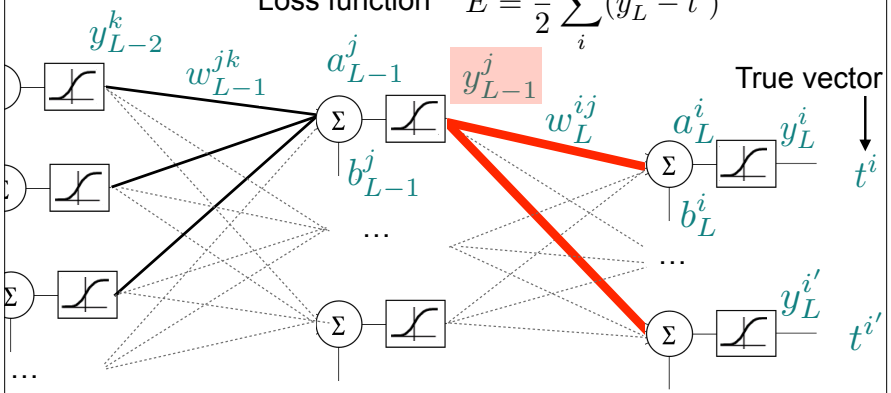
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial y_{L-1}^j}$$

Backprop with more than one node per layer

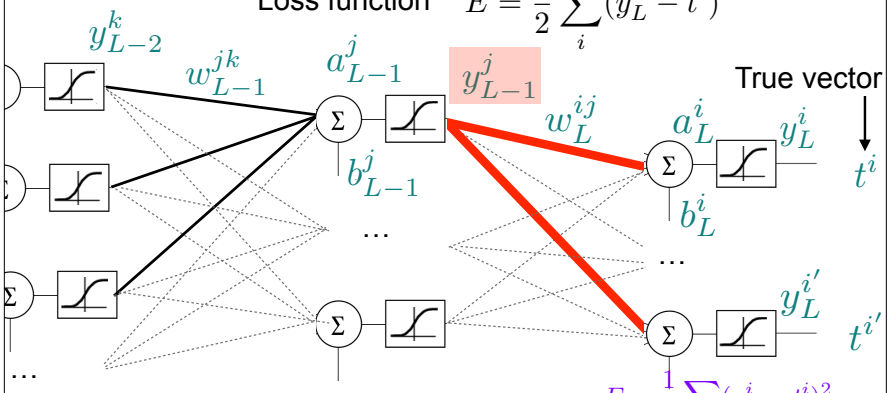
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial y_{L-1}^j}$$

Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



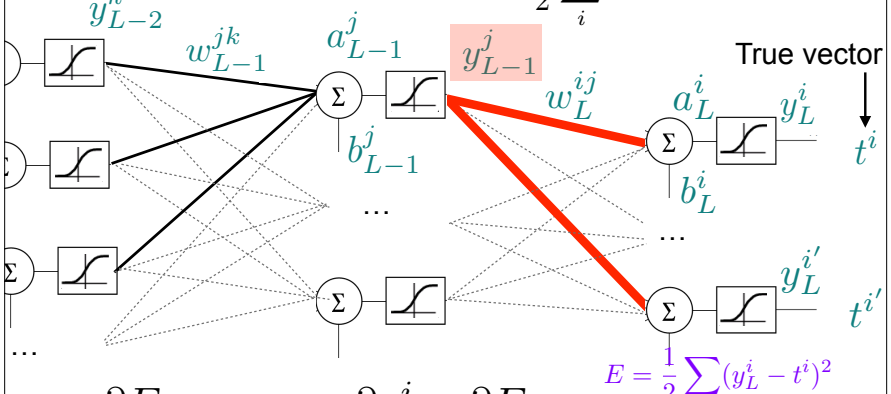
$$\frac{\partial E}{\partial y_{L-1}^j}$$

Take all a_L^i into consideration

$$E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$$
$$y_L^i = \sigma(a_L^i)$$
$$a_L^i = \sum_j w_L^{ij} y_{L-1}^j + b_L^i$$

Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$

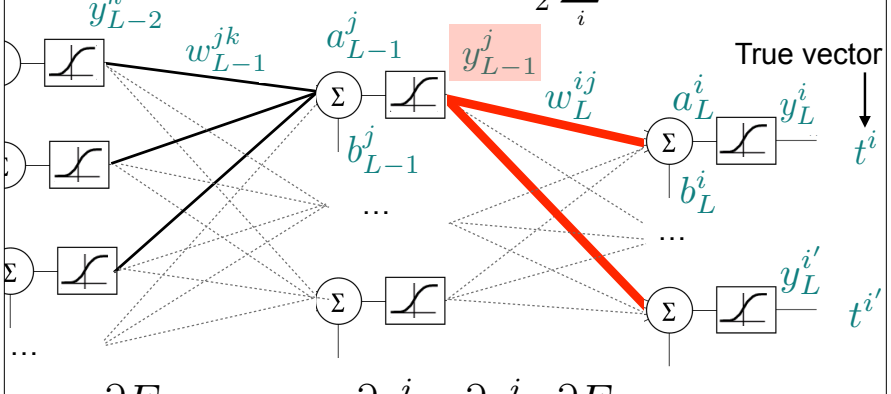


$$\frac{\partial E}{\partial y_{L-1}^j} = \sum_i \frac{\partial a_L^i}{\partial y_{L-1}^j} \frac{\partial E}{\partial a_L^i}$$

$$E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$$
$$y_L^i = \sigma(a_L^i)$$
$$a_L^i = \sum_j w_L^{ij} y_{L-1}^j + b_L^i$$

Backprop with more than one node per layer

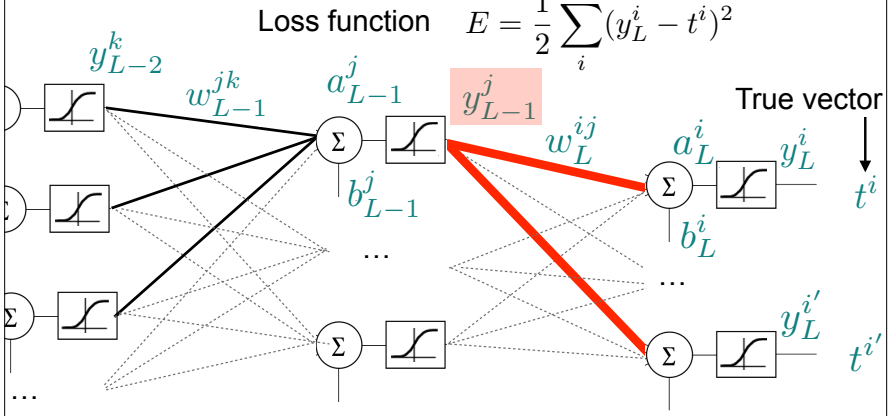
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial y_{L-1}^j} = \sum_i \frac{\partial a_L^i}{\partial y_{L-1}^j} \frac{\partial y_L^i}{\partial a_L^i} \frac{\partial E}{\partial y_L^i}$$

Backprop with more than one node per layer

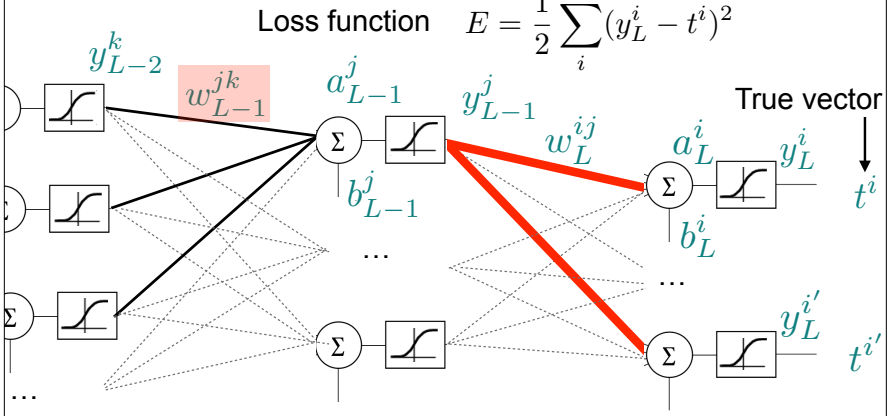
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial y_{L-1}^j} = \sum_i w_L^{ij} y_L^i (1 - y_L^i)(y_L^i - t^i)$$

Backprop with more than one node per layer

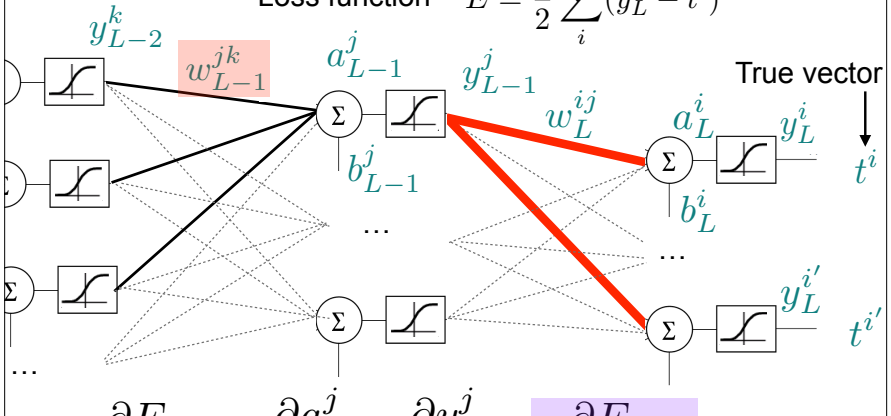
Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial w_{L-1}^{jk}}$$

Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$

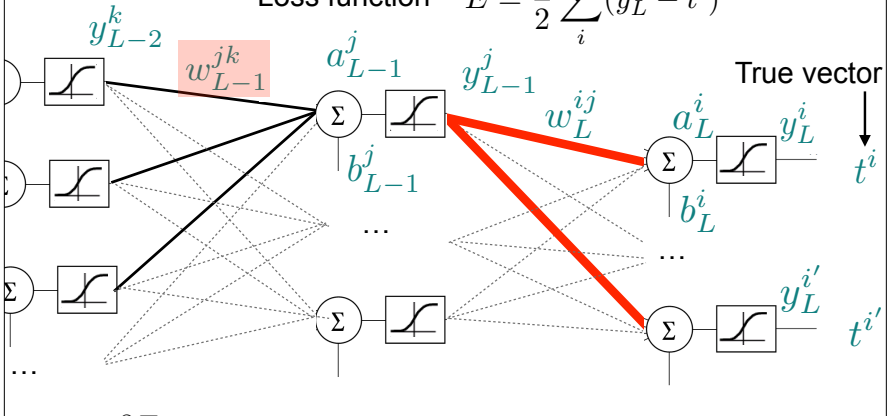


$$\frac{\partial E}{\partial w_{L-1}^{jk}} = \frac{\partial a_{L-1}^j}{\partial w_{L-1}^{jk}} \frac{\partial y_{L-1}^j}{\partial a_{L-1}^j} \frac{\partial E}{\partial y_{L-1}^j}$$

Already Computed

Backprop with more than one node per layer

Loss function $E = \frac{1}{2}\sum_i (y_L^i - t^i)^2$



$$\frac{\partial E}{\partial w_{L-1}^{jk}} = y_{L-2}^k y_{L-1}^j (1 - y_{L-1}^j) \sum_i w_L^{ij} y_L^i (1 - y_L^i)(y_L^i - t^i)$$

How to train

Random initialization, then repeat:

- Sample a training example
  - Input training example and compute all outputs y from all nodes

How to train

Random initialization, then repeat:

- Sample a training example
  - Input training example and compute all outputs y from all nodes
  - Back-propagate E (weighting it by the gradient of previous layer and activation function), i.e. compute $\frac{\partial E}{\partial y_H^i}$ for all layers stating from output layer L to first layer 1.

How to train

Random initialization, then repeat:

- Sample a training example
  - Input training example and compute all outputs y from all nodes
  - Back-propagate E (weighting it by the gradient of previous layer and activation function), i.e. compute $\frac{\partial E}{\partial y_H^i}$ for all layers stating from output layer L to first layer 1.
  - Calculate the gradients $\frac{\partial E}{\partial w_H^{ij}}$ and $\frac{\partial E}{\partial b_H^i}$

How to train

Random initialization, then repeat:

- Sample a training example
  - Input training example and compute all outputs y from all nodes
  - Back-propagate E (weighting it by the gradient of previous layer and activation function), i.e. compute $\frac{\partial E}{\partial y_H^i}$ for all layers stating from output layer L to first layer 1.
  - Calculate the gradients $\frac{\partial E}{\partial w_H^{ij}}$ and $\frac{\partial E}{\partial b_H^i}$
  - Update the parameters

$$w_H^{ij} \leftarrow w_H^{ij} - \eta \frac{\partial E}{\partial w_H^{ij}} \qquad b_H^i \leftarrow b_H^i - \eta \frac{\partial E}{\partial b_H^i}$$

How to train

Random initialization, then repeat:

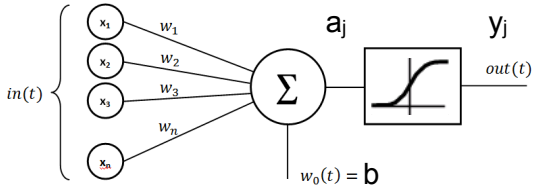
- Sample a training example
  - Input training example and compute all outputs y from all nodes
  - Back-propagate E (weighting it by the gradient of previous layer and activation function), i.e. compute $\frac{\partial E}{\partial y_H^i}$ for all layers stating from output layer L to first layer 1.
  - Calculate the gradients $\frac{\partial E}{\partial w_H^{ij}}$ and $\frac{\partial E}{\partial b_H^i}$
  - Update the parameters

$$w_H^{ij} \leftarrow w_H^{ij} - \eta \frac{\partial E}{\partial w_H^{ij}} \qquad b_H^i \leftarrow b_H^i - \eta \frac{\partial E}{\partial b_H^i}$$

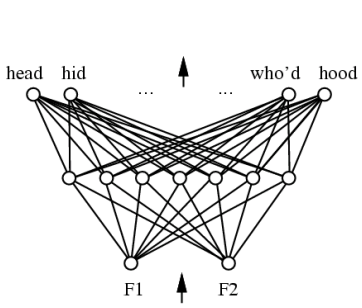
but… function not convex…
still gradient descent could lead to "good" solutions

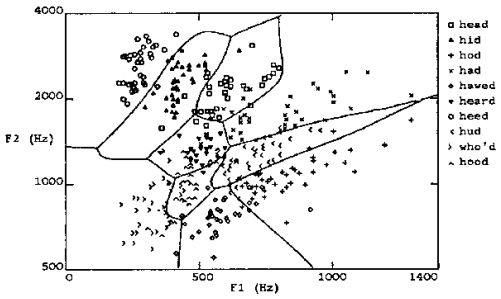
Questions

- Is there a difference between LR and one sigmoid unit?

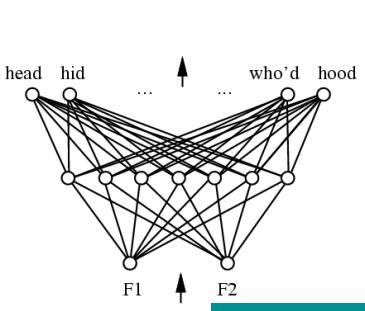


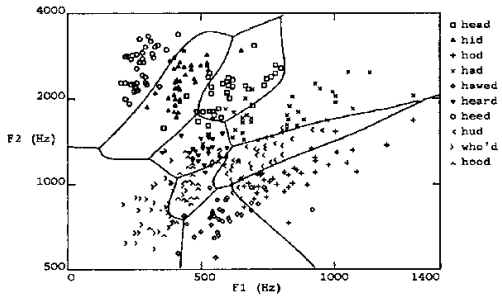
Multilayer Networks of Sigmoid Units





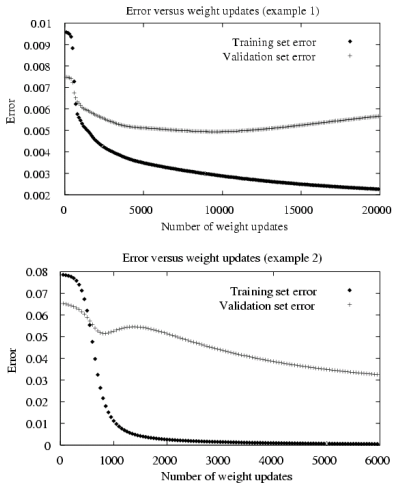
Multilayer Networks of Sigmoid Units





Can learn highly non-linear decision surfaces. However, the function that a deep neural net learns is typically difficult to interpret

Overfitting in ANNs



Regularization

- To avoid overfitting, add penalty, e.g. L2 penalty:

$$\arg\min_W \sum_l ||\hat{f}(\mathbf{x}^l) - \mathbf{t}^l||_2^2$$

$$\arg\min_W \sum_l ||\hat{f}(\mathbf{x}^l) - \mathbf{t}^l||_2^2 + \lambda||W||_2^2$$

Regularization

- To avoid overfitting, add penalty, e.g. L2 penalty:

$$\arg\min_W \sum_l ||\hat{f}(\mathbf{x}^l) - \mathbf{t}^l||_2^2$$

$$\arg\min_W \sum_l ||\hat{f}(\mathbf{x}^l) - \mathbf{t}^l||_2^2 + \lambda||W||_2^2$$

- Early stopping: stop when validation performance worsens. This has a regularization effect

Regularization

- To avoid overfitting, add penalty, e.g. L2 penalty:

$$\arg\min_W \sum_l ||\hat{f}(\mathbf{x}^l) - \mathbf{t}^l||_2^2$$

$$\arg\min_W \sum_l ||\hat{f}(\mathbf{x}^l) - \mathbf{t}^l||_2^2 + \lambda||W||_2^2$$

- Early stopping: stop when validation performance worsens. This has a regularization effect
- Dropout: randomly ignore a proportion of nodes p during training (at each iteration). Account for this during testing (multiplying by p) or by scaling inversely during training (dividing by p).

How to train

- What happens if we initialize weights to 0?

How to train

- What happens if we initialize weights to 0?
- Random initialization:
  - Different results (we might learn the same function but at different nodes, or learn different functions since the function is not convex)
  - Repeat multiple times and pick the one with best cross-validation performance

Many hyperparameters

- Many knobs to tune:
  - Structure of network
    - Number of layers
    - Number of nodes per layer
    - Other: recurrence, convolution
  - Learning rate
  - Initialization
  - Loss function
  - Gate
  - Stopping criterion (early stopping?)

Many hyperparameters

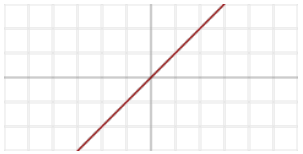
- Many knobs to tune:
  - Structure of network
    - Number of layers
    - Number of nodes per layer
    - Other: recurrence, convolution
  - Learning rate
  - Initialization
  - Loss function
  - Gate
  - Stopping criterion

Many additional tweaks that can help

- Batch normalization
- Learning rate decay
- Momentum
- …

Choice of activation gate

Linear

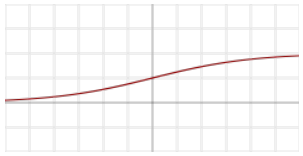


$f(x) = x$ $\quad$ $f'(x) = 1$

Can be used to predict continuous values at output.
What happens when you stay linear layers?

Choice of activation gate

Sigmoid

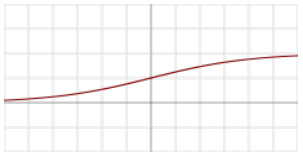


$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x)(1-f(x))$$

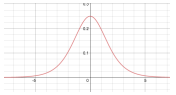
Outputs between 0 and 1, can be used for probability
Can saturate when very low or very high weights
Contributes to vanishing gradient

Choice of activation gate

Sigmoid



f'(x)

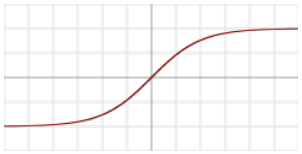


$$f(x) = \sigma(x) = \frac{1}{1+e^{-x}} \qquad f'(x) = f(x)(1-f(x))$$

Outputs between 0 and 1, can be used for probability
Can saturate
Contributes to vanishing gradient

Choice of activation gate

tanh

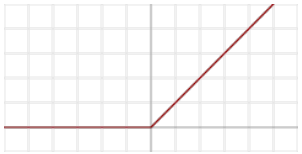


$$f(x) = \tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})} \qquad f'(x) = 1 - f(x)^2$$

Range -1 to 1

Choice of activation gate

Rectified Linear Unit (ReLu)

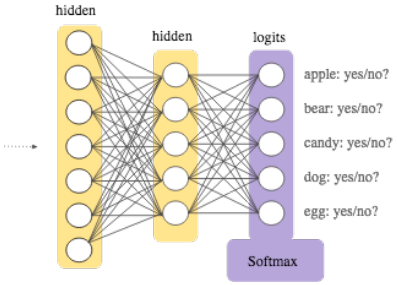


$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

Solve the vanishing gradient problem, but have a problem that nodes might die when negative value and never update. Can fix with leaky ReLu

Choice of activation gate

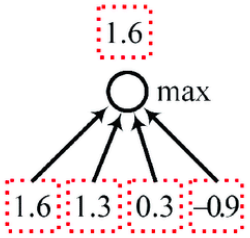
Softmax



$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \qquad \frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$$

Choice of activation gate

Maxpool



$$f(\vec{x}) = \max_i x_i \qquad \frac{\partial f}{\partial x_j} = \begin{cases} 1 & \text{for } j = \operatorname{argmax}_i x_i \\ 0 & \text{for } j \neq \operatorname{argmax}_i x_i \end{cases}$$

Choice of loss function

MSE

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

L2

$$\mathcal{L} = \sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2$$

Choice of loss function

Binary cross-entropy

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Multiclass:

$$-\frac{1}{n}\sum_{n}\sum_{k} y_{nk}\log f_k(\boldsymbol{x}_n)$$

$$\mathcal{L} = y\log(\sigma(\mathbf{z})) + (1-y)\log(1-\sigma(\mathbf{z})),$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = (y - \sigma(\mathbf{z}))\cdot\mathbf{x}$$

Choice of loss function

Binary cross-entropy

$$\mathcal{L} = -\frac{1}{n}\sum_{i=1}^{n}\left[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})\right]$$

Doesn't have saturation problem

$$\mathcal{L} = y\log(\sigma(\mathbf{z})) + (1-y)\log(1-\sigma(\mathbf{z})),$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = (y - \sigma(\mathbf{z}))\cdot\mathbf{x}$$

Deep Networks

Deep networks: informal term for more recent generation of neural nets, with features such as:

- more hidden layers
- built from more heterogenous units
  - sigmoid, rectilinear, max pooling, LSTM, …
- shared weights across units (convolutional)
- with application-specific network architecture
  - time series, computer vision, speech recognition, ...
  - recurrent networks, max-pooled convolutional layers with local receptive fields...
- bi-directional units
- pretrained on unlabelled data (auto-encoders)
- ...

Impact of Deep Learning

- Speech Recognition
- Computer Vision
- Recommender Systems
- Language Understanding
- Drug Discovery and Medical Image Analysis

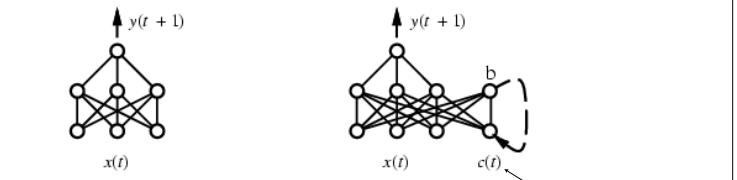
[Courtesy of R. Salakhutdinov]

Training Networks on Time Series

- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns
  - e.g., anticipate the next word in the sentence

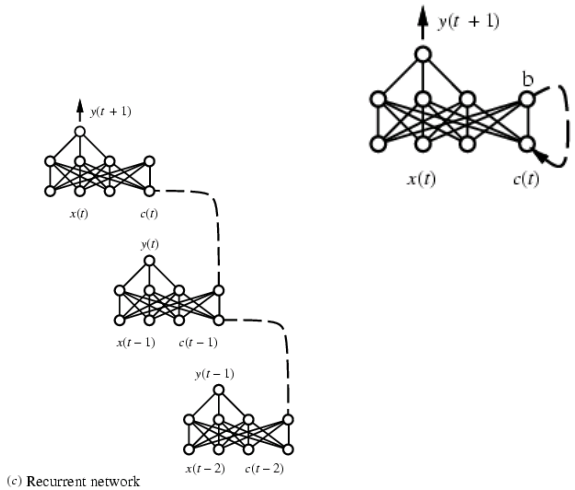
Recurrent Networks: Time Series

- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns
- Idea: use hidden layer in network to capture/ remember state history



(a) Feedforward network (b) Recurrent network

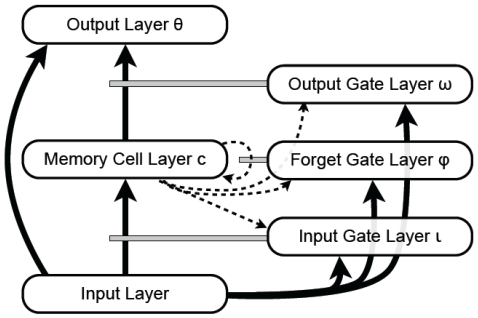
Recurrent Networks on Time Series



(c) Recurrent network unfolded in time

Long-Short Term Memory (LSTM) Units

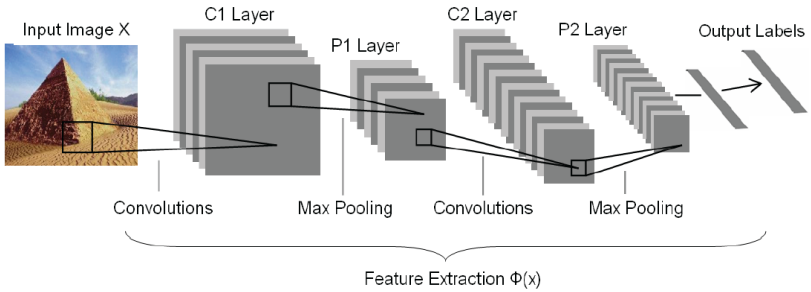
- Threshold unit/subnetwork with memory
  - still trainable with gradient descent



LSTM-g unit from [Monner & Regia, 2013]

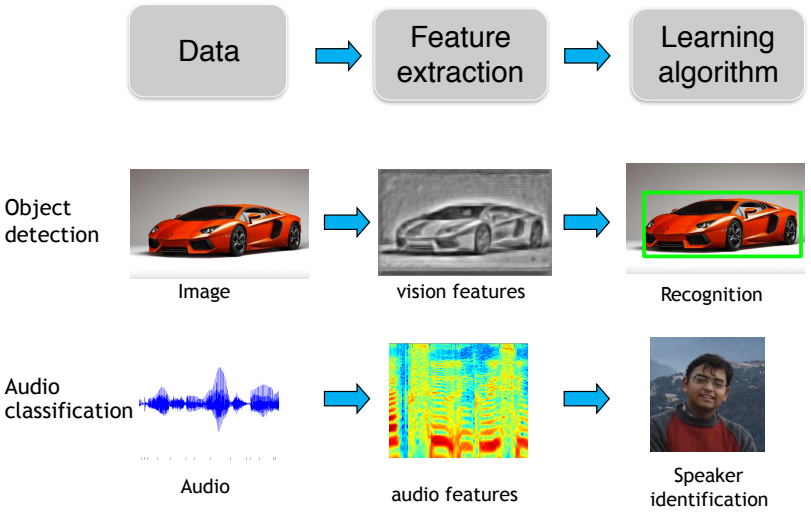
Convolutional Neural Nets for Image Recognition

[Le Cun, 1992]



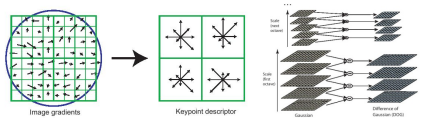
- specialized architecture: mix different types of units, not completely connected, motivated by primate visual cortex
- many shared parameters, stochastic gradient training
- very successful! now many specialized architectures for vision, speech, translation, …

Feature Representations: Traditionally

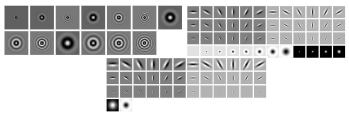


[Courtesy of R. Salakhutdinov]

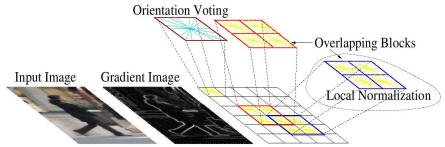
Computer Vision Features



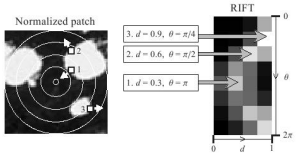
SIFT



Textons

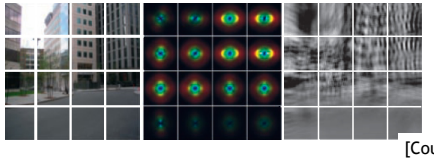


HoG



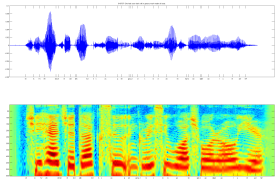
RIFT

GIST

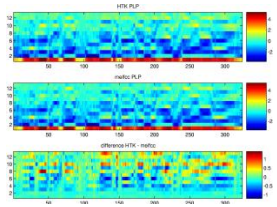


[Courtesy, R. Salakhutdinov]

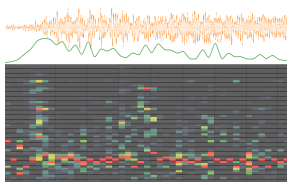
Audio Features



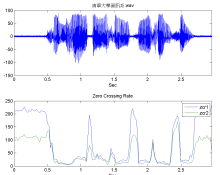
Spectrogram



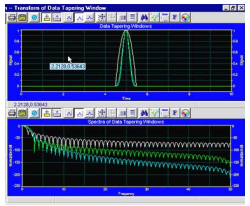
MFCC



Flux



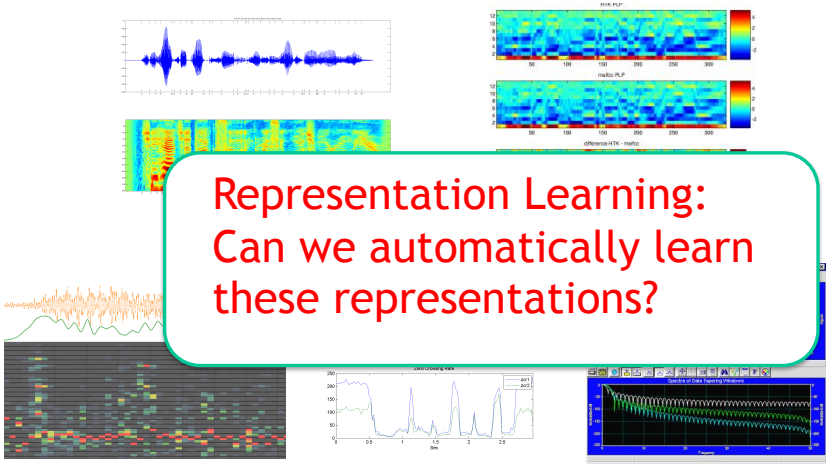
ZCR



Rolloff

[Courtesy, R. Salakhutdinov]

Audio Features



Representation Learning:
Can we automatically learn these representations?

Flux

ZCR

Rolloff

[Courtesy, R. Salakhutdinov]